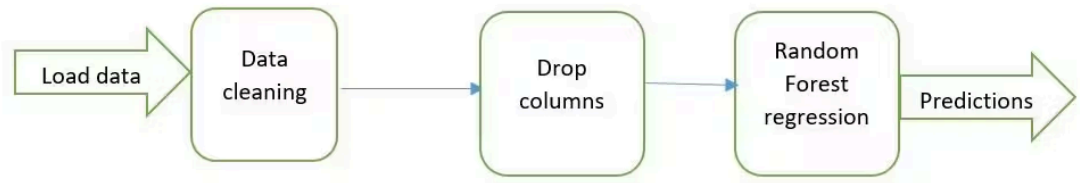
MLops Pipeline



Objective

The objective of this lab assessment is to evaluate your understanding of building and managing an MLOps pipeline.

Aim: To solve the MLOps pipeline problem using a regression model with the Bike Sharing dataset, we'll go through the following steps:

1. **Problem Definition**

Objective:
Predict the number of bike rentals (target variable: cnt) based on various features such as weather conditions, season, and time of day. The problem is a regression task where the goal is to predict a continuous variable.

Dataset:
The Bike Sharing dataset contains data related to bike rental counts with features like:
Open the link -> right-click on the download button -> Copy Link Address.

Description of the dataset: -

- season: Season of the year (1:winter, 2:spring, 3:summer, 4:fall).
- yr: Year (0: 2011, 1:2012).
- mnth: Month of the year (1 to 12).
- hr: Hour of the day (0 to 23).
- holiday: Whether the day is a holiday.
- weekday: Day of the week.
- workingday: Whether the day is a working day.
- weathersit: Weather situation (1: Clear, 2: Mist, 3: Light Snow/Rain, 4: Heavy Rain/Snow).
- temp: Normalized temperature in Celsius.
- hum: Normalized humidity.
- windspeed: Normalized wind speed.
- cnt: Count of total rental bikes (target variable).

2. **Initializing the Colab environment**
   Open your colab notebook -> connect runtime.
   In a new cell, type and run: -

```
!wget https://archive.ics.uci.edu/static/public/275/bike+sharing+dataset.zip
!unzip bike+sharing+dataset.zip
```

Wget is the non-interactive network downloader used to download files from the server even when the user has not logged on to the system. It can work in the background without hindering the current process.

Import the necessary packages:

```
import pandas as pd
```

3. Data Engineering

Data Collection:
Load the dataset into a Pandas DataFrame.

```
df = pd.read_csv('hour.csv')
df
```

Data Preprocessing:

- Handling Missing Values: Check for and handle any missing values.
- Feature Engineering: Create additional features if necessary (e.g., day/night feature based on hour).
- Normalization/Standardization: Normalize numerical features such as temp, hum, and windspeed.

```
df['day_night'] = df['hr'].apply(lambda x: 'day' if 6 <= x <= 18 else 'night')
Link to understand how to apply lambda function: Link

df.drop(['instant', 'casual', 'registered'], axis=1, inplace=True)
df['dteday'] = pd.to_datetime(df.dteday)
df['season'] = df.season.astype('category')
df['holiday'] = df.holiday.astype('category')
df['weekday'] = df.weekday.astype('category')
df['weathersit'] = df.weathersit.astype('category')
df['workingday'] = df.workingday.astype('category')
df['mnth'] = df.mnth.astype('category')
df['yr'] = df.yr.astype('category')
df['hr'] = df.hr.astype('category')
df.drop(columns=['dteday'], inplace=True)

# Separating features and target variable
X = df.drop(columns=['cnt'])  # Features
y = df['cnt']  # Target
```

Create Pipelines for Numerical and Categorical Features

The syntax of the pipeline is:

```
Pipeline(steps = [('step name', transform function), …])
```

For numerical features, we perform the following actions:

1. **SimpleImputer:** to fill in the missing values with the mean of that column.
2. **MinMaxScaler:** to scale the value to range from 0 to 1 (this will affect regression performance).

For categorical features, we perform the following actions:

1. **SimpleImputer:** to fill in the missing values with the most frequency value of that column.
2. **OneHotEncoder:** to split to many numerical columns for model training. (handle_unknown='ignore' is specified to prevent errors when it finds an unseen category in the test set)

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.pipeline import Pipeline

# Numerical features
numerical_features = ['temp', 'hum', 'windspeed']
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with mean
    ('scaler', MinMaxScaler())  # Normalize using MinMaxScaler
])

# Transforming above
X[numerical_features] = numerical_pipeline.fit_transform(X[numerical_features])

# Categorical features
categorical_features = ['season', 'weathersit', 'day_night']
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(sparse_output=False, drop='first'))
])

# Transforming above
X_encoded = categorical_pipeline.fit_transform(X[categorical_features])
# Converting it to a dataframe
```

```
X_encoded = pd.DataFrame(X_encoded,
columns=categorical_pipeline.named_steps['onehot'].get_feature_names_out(categorical_
features))

# Encoded categorical features + Numerical features
X = pd.concat([X.drop(columns=categorical_features), X_encoded], axis=1)
```

Split the data (20:80 ratio)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

4. Model Development

Model Selection:
A regression model like Linear Regression, Random Forest Regressor, or Gradient Boosting Regressor can be used.
To understand RandomForestRegressor, refer to the following: Link

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Visualize the importance of each feature: -

```
feat_importances = pd.DataFrame(model.feature_importances_, index=X_train.columns,
columns=["Importance"])
feat_importances.sort_values(by='Importance', ascending=False, inplace=True)

# Plotting each feature importance
feat_importances.plot(kind='bar', figsize=(12,8))
```

Model Evaluation:
Evaluate the model using metrics such as Mean Squared Error (MSE) and R-squared.

```
# Predictions
y_pred = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

ML Pipeline

```
final_pipeline = Pipeline([
    ('num_preprocess', numerical_pipeline),
    ('cat_preprocess', categorical_pipeline),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
])
```

```
set_config(display='diagram')# To display
final_pipeline
```

Finally, your output would be like this: